

Curso de introducción a MatLab

**PROGRAMA DE
INGENIERÍA ELÉCTRICA**

ALEXANDER MOLINA CABRERA

JHON EDUARD VALENCIA

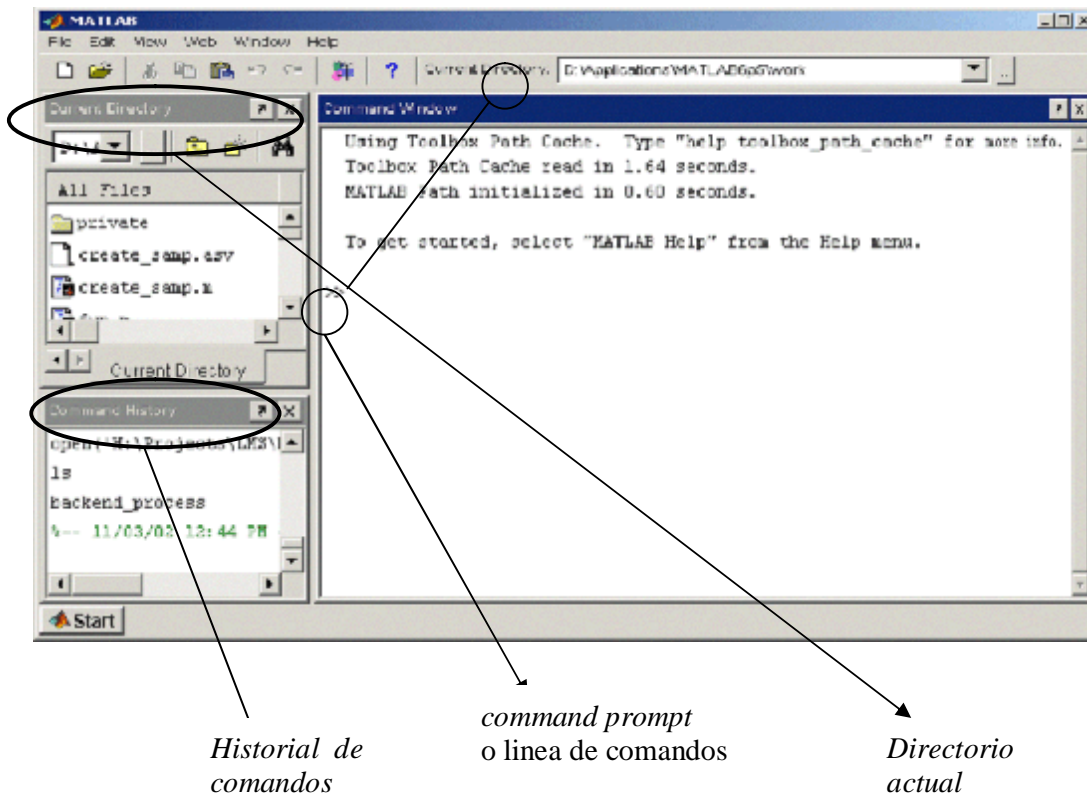
Introducción:

Este breve curso de introducción a MatLab[®], tiene como objeto familiarizar al estudiante de básicos de ingeniería eléctrica con una herramienta, que le permitirá realizar cálculos numéricos con matrices y vectores, al igual que con números escalares -tanto reales como complejos- que le serán de gran utilidad dentro de su proceso de formación como ingeniero electricista.

A grandes rasgos, los temas aquí introducidos son:

- § Generalidades.
- § Vectores y matrices.
- § Operaciones con vectores y matrices.
- § Variables lógicas.
- § Polinomios.
- § Derivadas y integrales.
- § Gráficas de funciones.
- § Programación con MATLAB.

GENERALIDADES



- § MATLAB distingue entre mayúsculas y minúsculas.
- § La comilla ' es la que, en un teclado estándar, se encuentra en la tecla de la interrogación.
- § Los comentarios deben ir precedidos por % o, lo que es lo mismo, MATLAB ignora todo lo que vaya precedido por el símbolo %.
- § La ayuda de MATLAB es bastante útil; para acceder a la misma basta teclear `help`. Es recomendable usarlo para obtener una información más precisa sobre la sintaxis y diversas posibilidades de uso de los comandos.
- § Si deseamos obtener ayuda sobre una función específica podemos teclear `help función` donde `función` representa el comando del sobre el cual deseamos ayuda por ejemplo, si digitamos `help help` obtendremos ayuda sobre el comando `ayuda`.
- § Si deseamos conocer la fecha del sistema podemos digitar `date`
- § Para salir de MatLab[®] podemos digitar `quit` o `exit`
- § Para limpiar el prompt podemos digitar `clc`

Los cálculos que no se asignan a una variable en concreto se asignan a la variable de respuesta por defecto que es `ans` (del inglés, *answer*):

```
>>2+3

ans =
5
```

Sin embargo, si el cálculo se asigna a una variable, el resultado queda guardado en ella:

```
>>x=2+3

x =
5
```

Para conocer el valor de una variable, basta teclear su nombre:

```
>>x

x =
5
```

Si se añade un punto y coma (;) al final de la instrucción, la máquina no muestra la respuesta...

```
>>y=5*4;
```

... pero no por ello deja de realizarse el cálculo.

```
>>y
```

```
y =  
20
```

Las operaciones se evalúan por orden de prioridad: primero las potencias, después las multiplicaciones y divisiones y, finalmente, las sumas y restas. Las operaciones de igual prioridad se evalúan de izquierda a derecha:

```
>>2/4*3
```

```
ans =  
1.5000
```

```
>>2/(4*3)
```

```
ans =  
0.1667
```

Se pueden utilizar las funciones matemáticas habituales. Así, por ejemplo, la función coseno,

```
>>cos(pi) % pi es una variable con valor predeterminado 3.14159...
```

```
ans =  
-1
```

o la función exponencial

```
>>exp(1) % Función exponencial evaluada en 1, es decir, el número e
```

```
ans =  
2.7183
```

Además de la variable `pi`, MATLAB tiene otras variables con valor predeterminado; éste se pierde si se les asigna otro valor distinto. Por ejemplo:

```
>>eps % Épsilon de la máquina. Obsérvese que MATLAB trabaja en  
doble precisión
```

```
ans =  
2.2204e-016
```

pero...

```
>>eps=7
```

```
eps =  
7
```

Otro ejemplo de función matemática: la raíz cuadrada; como puede verse, trabajar con complejos no da ningún tipo de problema. La unidad imaginaria se representa en MATLAB como `i` o `j`, variables con dicho valor como predeterminado:

```
>>sqrt(-4)
```

```
ans =  
0+ 2.0000i
```

El usuario puede controlar el número de decimales con que aparece en pantalla el valor de las variables, sin olvidar que ello no está relacionado con la precisión con la que se hacen los cálculos, sino con el aspecto con que éstos se muestran:

```
>>1/3
```

```
ans =  
0.3333
```

```
>>format long
```

```
>>1/3
```

```
ans =  
0.3333333333333333
```

```
>>format          % Vuelve al formato estándar que es el de 4 cifras  
decimales
```

Para conocer las variables que se han usado hasta el momento:

```
>>who
```

```
Your variables are:  
ans eps x y
```

o, si se quiere más información (obsérvese que todas las variables son *arrays*):

```
>>whos
```

Name	Size	Bytes	Class
ans	1x1	8	double array
eps	1x1	8	double array
x	1x1	8	double array
y	1x1	8	double array

```
Grand total is 4 elements using 32 bytes
```

Para deshacerse de una variable

```
>>clear y
```

```
>>who
```

```
Your variables are:  
ans eps x
```

VECTORES Y MATRICES.

Para definir un vector fila, basta introducir sus coordenadas entre corchetes:

```
>>v=[1 2 3]      % Vector de 3 coordenadas
```

```
v=  
1 2 3
```

```
>>w=[4 5 6];
```

El operador ' es el de trasposición (en realidad trasposición y conjugación):

```
>>w'
```

```
ans =  
4  
5  
6
```

Si queremos declarar un vector de coordenadas equiespaciadas entre dos dadas, por ejemplo, que la primera valga 0, la última 20 y la distancia entre coordenadas sea 2, basta poner:

```
>>vect1=0:2:20
```

```
vect1 =  
0 2 4 6 8 10 12 14 16 18 20
```

Equivalentemente, si lo que conocemos del vector es que la primera coordenada vale 0, la última 20 y que tiene 11 en total, escribiremos:

```
>>vect2=linspace(0,20,11)
```

```
vect2 =  
0 2 4 6 8 10 12 14 16 18 20
```

A las coordenadas de un vector se accede sin más que escribir el nombre del vector y, entre paréntesis, su índice:

```
>>vect2(3)
```

```
ans =  
4
```

y se pueden extraer subvectores, por ejemplo:

```
>>vect2(2:5)
```

```
ans=  
2 4 6 8
```

O,

```
>>vect1(:)
```

```
ans=  
0  
2  
4  
6  
8  
10  
12  
14  
16  
18  
20
```

Las matrices se escriben como los vectores, pero separando las filas mediante un punto y coma; así una matriz 3x3:

```
>>M=[1 2 3;4 5 6;7 8 9]
```

```
M =  
1 2 3  
4 5 6  
7 8 9
```

```
>>M' % Su traspuesta (su adjunta)
```

```
ans =  
1 4 7  
2 5 8  
3 6 9
```

```
>>mat=[v;i;0 0 1] % También es una matriz 3x3
```

```
mat =  
1 2 3  
4 5 6  
0 0 1
```

A los elementos de una matriz se accede sin más que escribir el nombre de la matriz y, entre paréntesis, los respectivos índices:

```
>>mat(1,3)    % Elemento en la primera fila y tercera columna de la
matriz mat
```

```
ans =
3
```

También se puede acceder a un fila o columna completas,

```
>>mat(:,2)    % Segunda columna de mat
```

```
ans =
2
5
0
```

```
>>mat(2,:)    % Su segunda fila
```

```
ans =
4 5 6
```

acceder a la matriz como si fuera una columna,

```
>>M(2:7)      % Los elementos segundo a séptimo de la matriz como
columna
```

```
ans =
4
7
2
5
8
3
```

o acceder a cualquiera de sus submatrices

```
>>mat(2:3,[1 3]) % Submatriz formada por los elementos que están en
% "todas" las filas que hay entre la segunda y la
tercera y
% en las columnas primera y tercera
```

```
ans =
4 6
0 1
```

Existen algunas matrices definidas previamente; por ejemplo, la matriz identidad,

```
>>eye(5)      % eye se pronuncia en inglés como I
```

```
ans =
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

la matriz nula,

```
>>zeros(3)
```

```
ans =  
0 0 0  
0 0 0  
0 0 0
```

o la matriz cuyos elementos valen todos 1:

```
>>ones(4)
```

```
ans =  
1 1 1 1  
1 1 1 1  
1 1 1 1  
1 1 1 1
```

Se puede conocer el tamaño de una matriz y la longitud de un vector:

```
>>size(mat) % Dimensiones de la matriz mat (número de filas y de  
columnas)
```

```
ans =  
3 3
```

```
>>size(v)
```

```
ans =  
1 3
```

```
>>length(v) % Longitud del vector (número de coordenadas)
```

```
ans =  
3
```

Existen comandos que permiten crear de forma sencilla matrices. Por ejemplo:

```
>>diag(v) % Matriz diagonal cuya diagonal es el vector v
```

```
ans =  
1 0 0  
0 2 0  
0 0 3
```

```
>>diag(diag(M)) % Matriz diagonal con la diagonal de M. La sentencia  
diag(M) da  
% el vector formado por la diagonal de la matriz M
```

```
ans =  
1 0 0  
0 5 0  
0 0 9
```

```

>>diag(ones(1,4),1)+diag(ones(1,4),-1) % Matriz tridiagonal 5x5 con 0
en la diagonal                                % principal y 1 en la sub y
superdiagonal

ans =
0 1 0 0 0
1 0 1 0 0
0 1 0 1 0
0 0 1 0 1
0 0 0 1 0

>>tril(M) % Matriz formada por la parte triangular inferior de M.

ans =
1 0 0
4 5 0
7 8 9

>>triu(M) % Matriz formada por la parte triangular superior de M.

ans =
1 2 3
0 5 6
0 0 9

```

OPERACIONES CON VECTORES Y MATRICES.

Las funciones matemáticas elementales están definidas de forma que se pueden aplicar sobre *arrays*. El resultado es el *array* formado por la aplicación de la función a cada elemento del *array*. Así:

```

>>log(v)

ans =
0 0.6931 1.0986

>>p=(0:0.1:1)*pi % Vector definido como el producto de un vector por
un escalar

p =
Columns 1 through 7
0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
Columns 8 through 11
2.1991 2.5133 2.8274 3.1416

>>x=sin(p)

x =
Columns 1 through 7
0 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511

```

```
Columns 8 through 11
0.8090 0.5878 0.3090 0.0000
```

Las operaciones habituales entre *arrays* (suma, resta y producto escalar de vectores; suma, resta, producto y potencia de matrices) se representan con los operadores habituales:

```
>>v,w      % Recordamos los valores de v y w

v =
1 2 3
w =
4 5 6

>>z=v*w'   % Producto escalar (producto de matrices 1x3 por 3x1)

z =
32

>>Z=w'*v   % Producto de matrices 3x1 por 1x3 = Matriz 3x3

Z =
4 8 12
5 10 15
6 12 18

>>v*w      % Los vectores v y w no se pueden multiplicar

??? Error using ==> *
Inner matrix dimensions must agree.

>>mat      % Recordamos el valor de la matriz mat

mat =
1 2 3
4 5 6
0 0 1

>>mat^2    % Matriz mat elevada al cuadrado

ans =
9 12 18
24 33 48
0 0 1
```

También pueden efectuarse multiplicaciones, divisiones y potencias de *arrays*, entendiéndolas como elemento a elemento (como, de hecho, se realizan la suma y la resta). El operador utilizado para ellas es el habitual precedido por un punto; es decir:

```
>>v.*w % Vector formado por los productos de las respectivas
coordenadas:
      % ans(i)=v(i)*w(i)
```

```

ans =
4 10 18

>>w./v % Vector formado por el cociente de cada coordenada de w entre
la
      % coordenada correspondiente de v: ans(i)=w(i)/v(i)

ans =
4.0000 2.5000 2.0000

>>mat.^2 % Matriz cuyos elementos son los de mat elevados
      % al cuadrado: ans(i,j)=mat(i,j)^2

ans =
 1  4  9
16 25 36
 0  0  1

```

Finalmente, pueden calcularse determinantes:

```

>>det(mat)

ans =
-3

```

y resolverse sistemas de ecuaciones lineales con el versátil comando \:

```

>>mat\v'

ans =
2.6667
-5.3333
3.000

```

VARIABLES LÓGICAS.

También existen variables lógicas que toman los valores 0 (falso) o 1 (verdadero) . Por ejemplo:

```

>>abs(v)>=2 % Vector lógico cuyas coordenadas valen 1 si la
coordenada
      % correspondiente de v es >= 2 y 0 si no lo es

ans =
0 1 1

>>vector=v(abs(v)>=2) % Vector formado por la coordenadas de v que
      % verifican la desigualdad

```

```

vector =
2 3

>>v2=[3 2 1]

v2 =
3 2 1

>>logica=v==v2    % Asignación de un valor lógico (el doble signo igual
es el
                    % igual lógico)

logica =
0 1 0

>>logic2=v~=v2    % Distinto (~ es el operador de negación)

logic2 =
1 0 1

```

POLINOMIOS.

Se puede trabajar con polinomios: basta tener en cuenta que un polinomio no es más que un vector. El orden de los coeficientes es de mayor a menor grado, por ejemplo:

```

>>p=[1 0 2 0 3]    % Polinomio  $x^4+2x^2+3$ 

p =
1 0 2 0 3

>>q=[2 1 0]        % Polinomio  $2x^2+x$ 

q =
2 1 0

```

MATLAB tiene funciones específicas para polinomios como:

```

>>polyval(p,-1)    % Evaluación del polinomio  $x^4+2x^2+3$  en  $x=-1$ 

ans =
6

>>pro=conv(p,q)    % Producto de los polinomios p y q

pro =
2 1 4 2 6 3 0

>>deconv(pro,p)    % Cociente entre pro y p; obviamente el resultado es
q

```

```

ans =
2 1 0

>>roots(pro)      % Raíces del polinomio pro

ans =
    0
 0.6050+1.1688i
 0.6050-1.1688i
-0.6050+1.1688i
-0.6050-1.1688i
-0.5000

>>poly([i -i 1/2 pi]) % Polinomio mónico que tiene por raíces a los
                        % números i, -i, 0.5 y pi

ans =
1.0000 -3.6416 2.5708 -3.6416 1.5708

```

DERIVADAS E INTEGRALES.

Dentro del módulo (*toolbox*) de matemática simbólica, se utiliza el programa de cálculo simbólico MAPLE. Con estas herramientas, se puede trabajar con funciones,

```

>>f='sin(x)'      % Función sin(x) definida mediante una cadena de
caracteres

```

```

f =
sin(x)

```

calcular derivadas,

```

>>diff(f)

```

```

ans =
cos(x)

```

```

>>diff(f,2)      % Derivada segunda de f

```

```

ans =
-sin(x)

```

o encontrar integrales.

```

>>int('log(x)') % Integral de la función logaritmo

```

```

ans =
x*log(x)-x

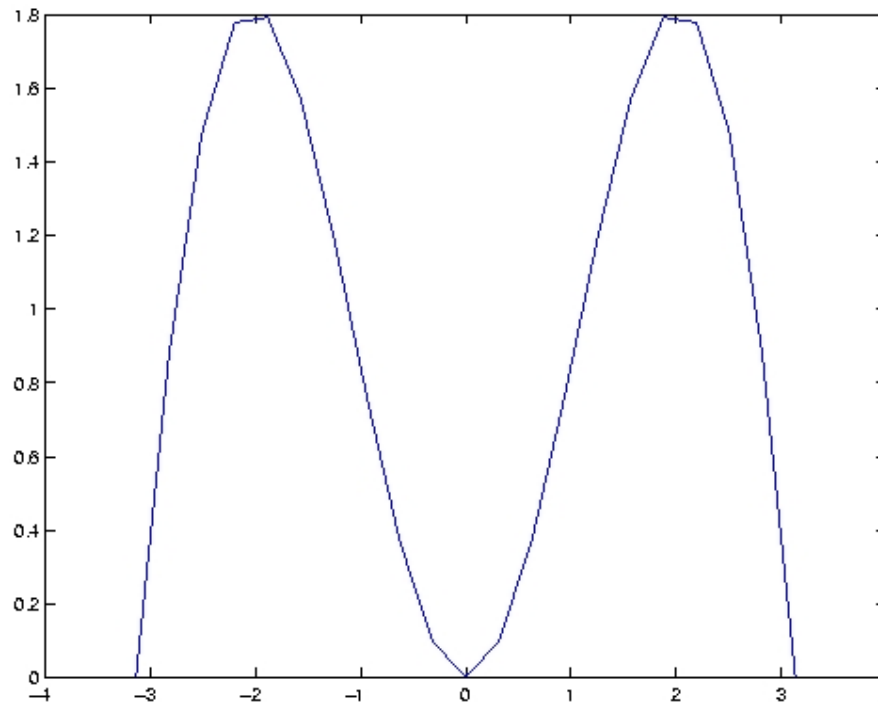
```

```
>>diff('x*log(x)-x') % Comprobación  
  
ans =  
log(x)
```

GRÁFICAS DE FUNCIONES.

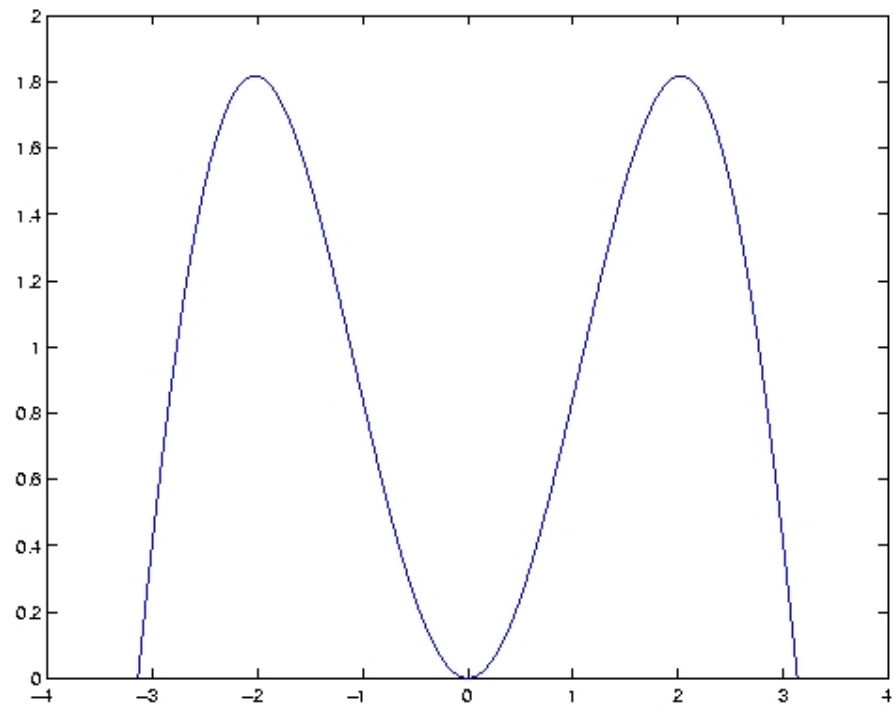
MATLAB tiene un gran potencial de herramientas gráficas. Se pueden dibujar los valores de un vector frente a otro (de la misma longitud):

```
>>x=pi*(-1:0.1:1);  
>>y=x.*sin(x);  
>>plot(x,y) % Por defecto une los puntos (x(i),y(i)) mediante  
una poligonal
```



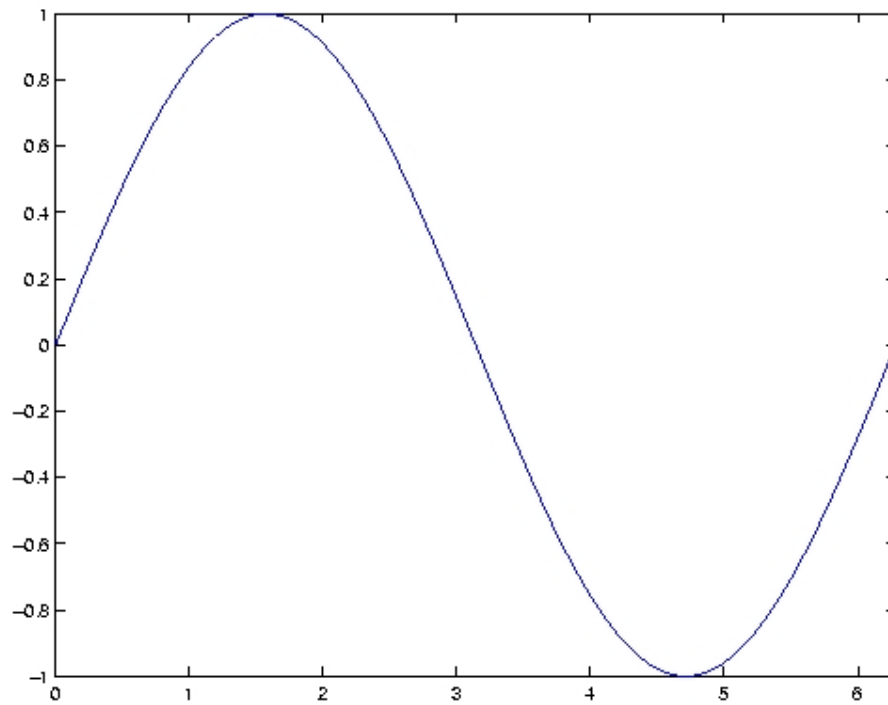
Como se ve, con pocos puntos la gráfica tiene un aspecto demasiado lineal a trozos. Para "engañar" al ojo, basta tomar más puntos.

```
>>x=pi*(-1:0.01:1);  
>>y=x.*sin(x);  
>>plot(x,y)
```



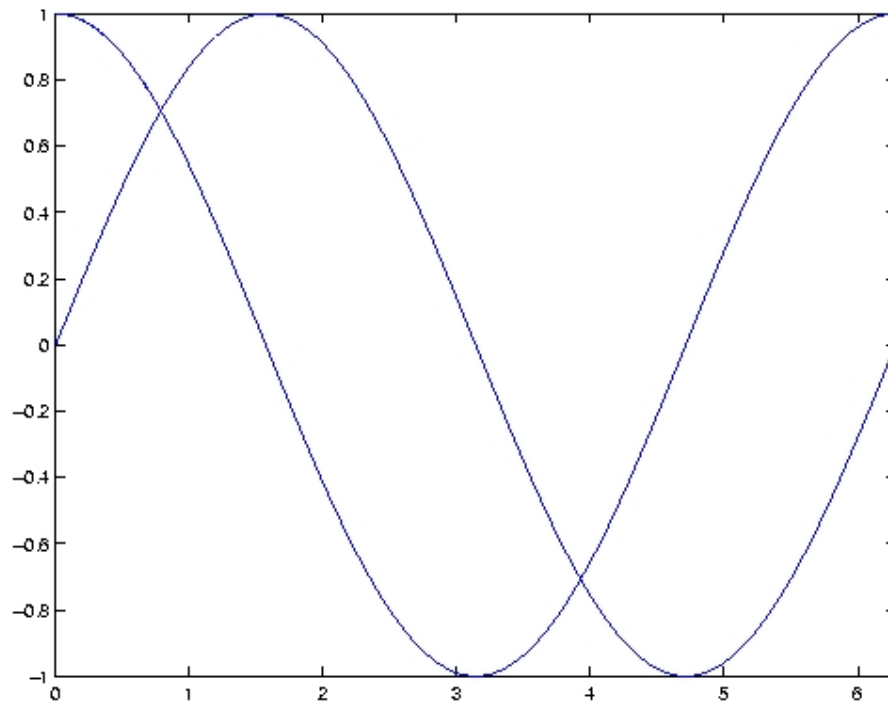
También pueden dibujarse funciones. Así:

```
>>fplot('sin(x)',[0 2*pi])    % Dibuja la función seno en el intervalo  
[0,2*pi]
```

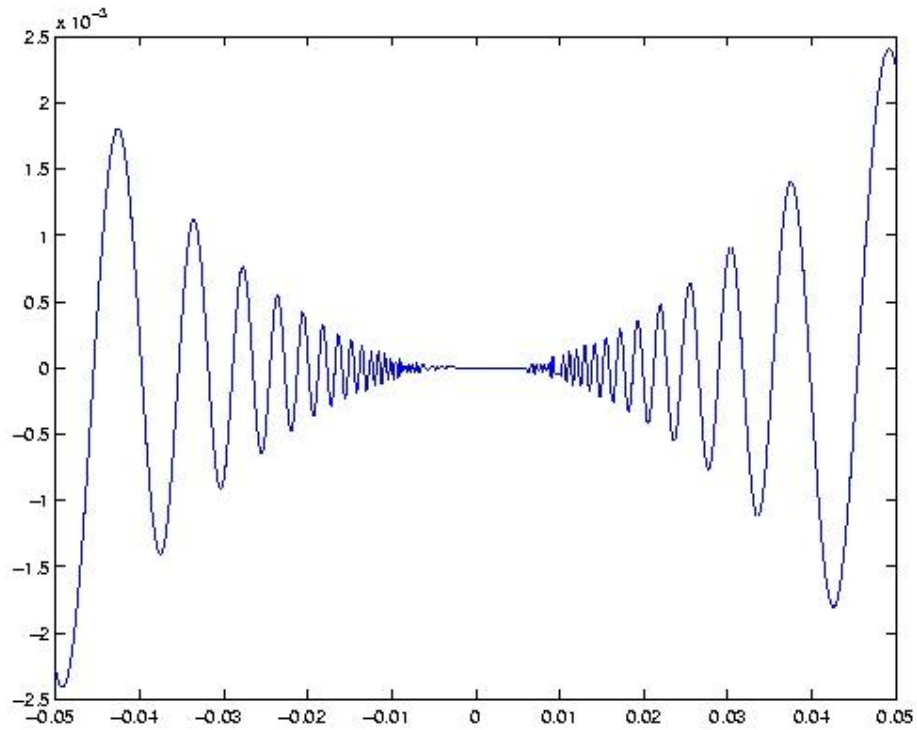


```
>>hold on % Mantiene en la ventana gráfica los dibujos  
anteriores
```

```
>>fplot('cos(x)',[0 2*pi]) % Dibuja sobre la gráfica anterior la  
función cos(x)
```

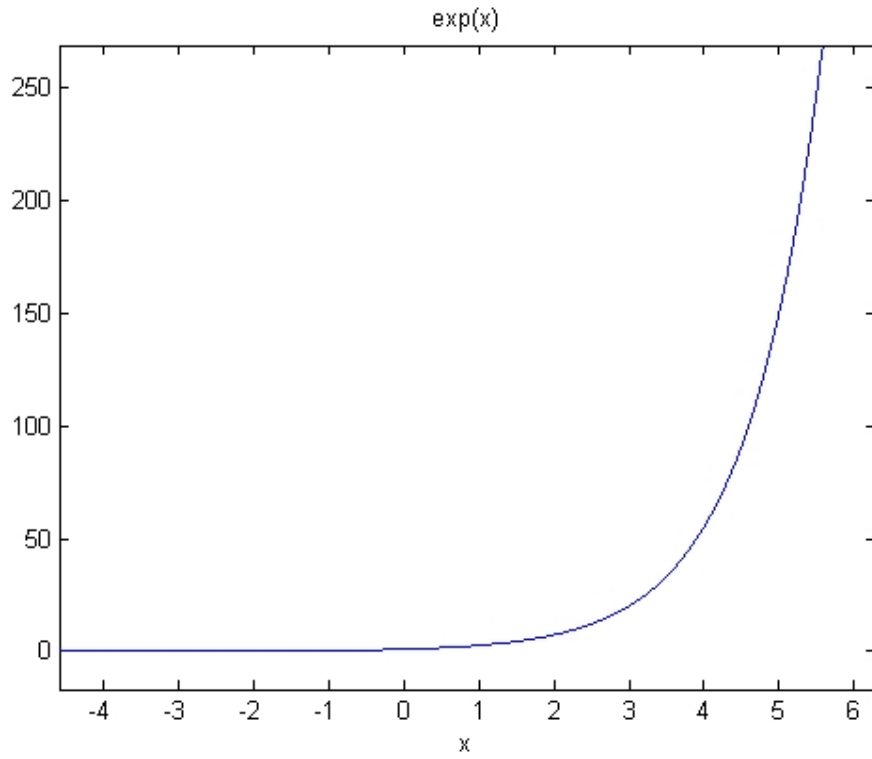


```
>>hold off                                % Con esto olvida los dibujos  
anteriores                                % y dibuja en una ventana nueva  
  
>>fplot('x^2*sin(1/x)',[-0.05 0.05]) % Dibuja la función x^2*sin(1/x)
```



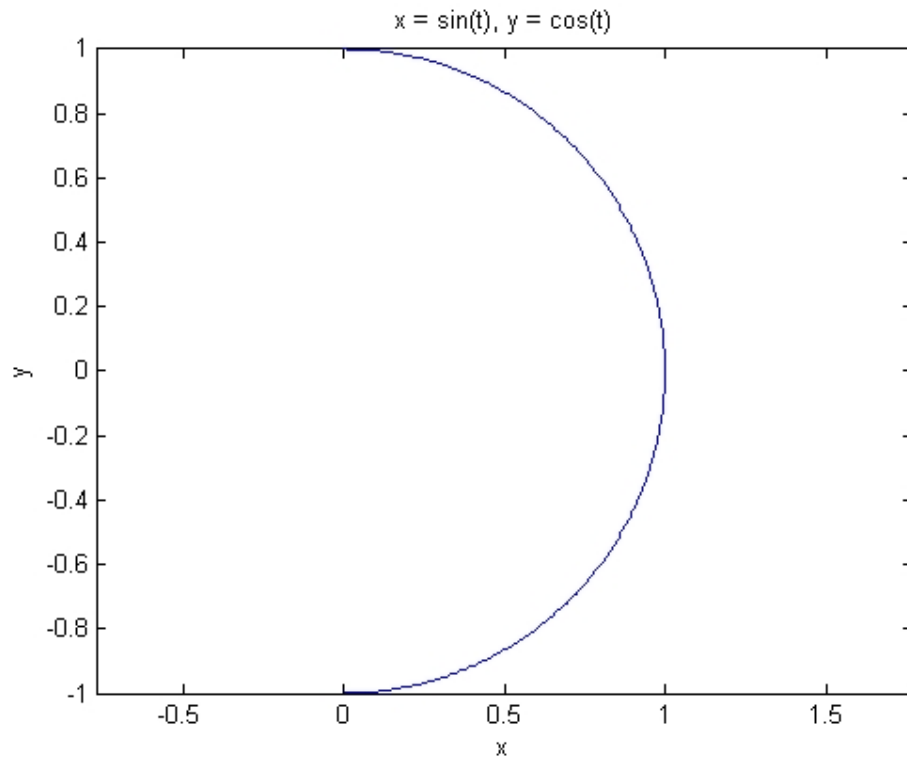
También puede usarse el versátil comando `ezplot` (se lee como *easy plot*) que permite dibujar funciones,

```
>>ezplot('exp(x)') % Dibuja la función exponencial en un intervalo  
adecuado a la función
```



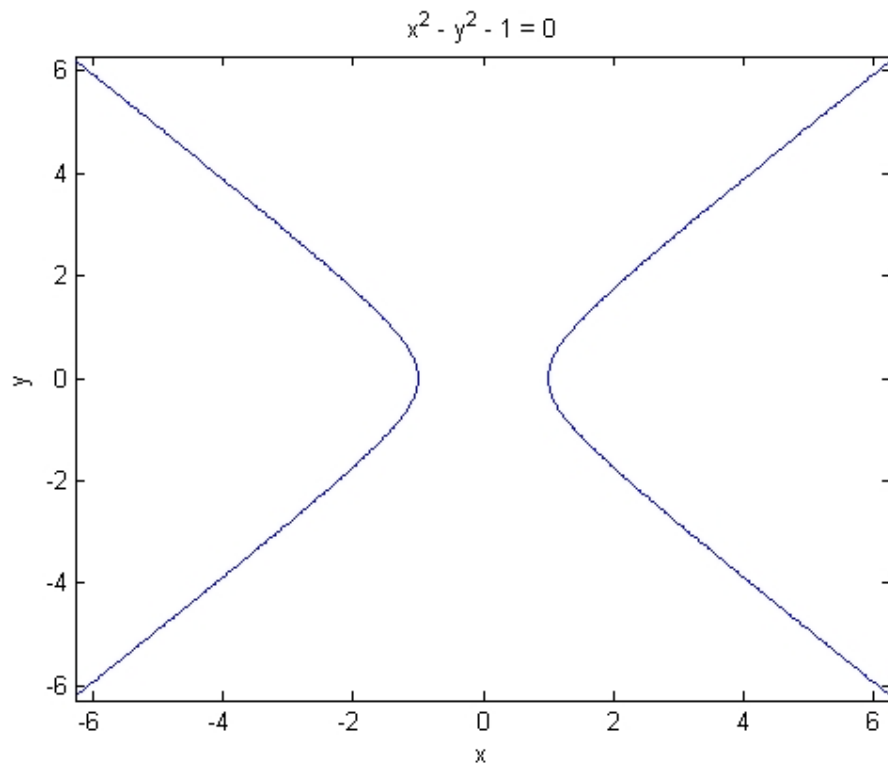
curvas en paramétricas,

```
>>ezplot('sin(t)','cos(t)',[0 pi])
```



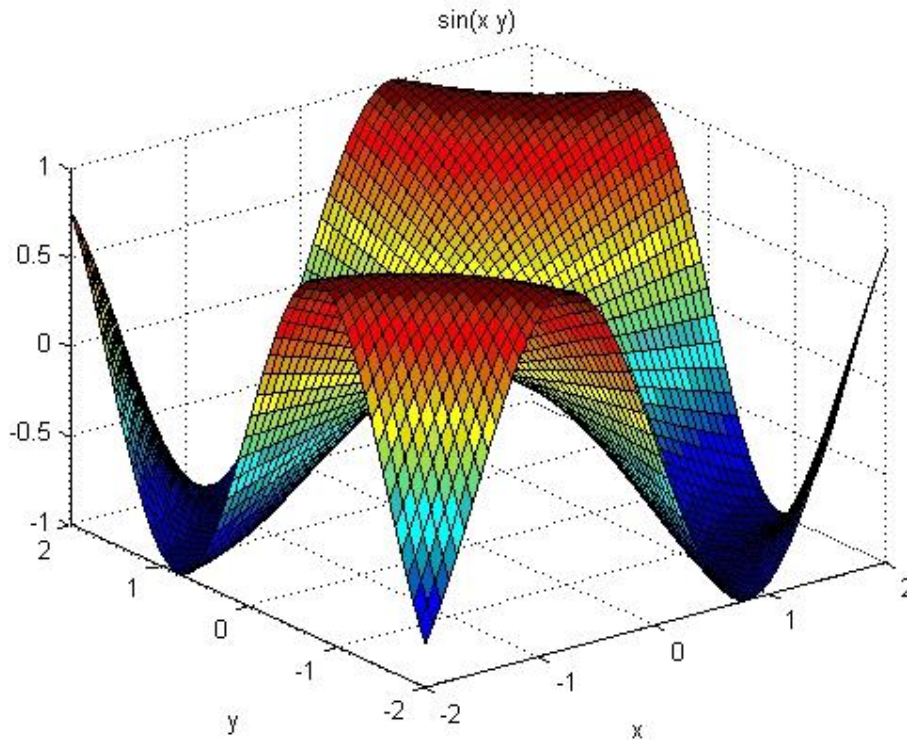
e implícitas

```
>>ezplot('x^2 - y^2 - 1')
```



También permite dibujar superficies. La forma más sencilla es mediante el comando `ezsurf`,

```
>>ezsurf('sin(x*y)',[-2 2 -2 2])
```

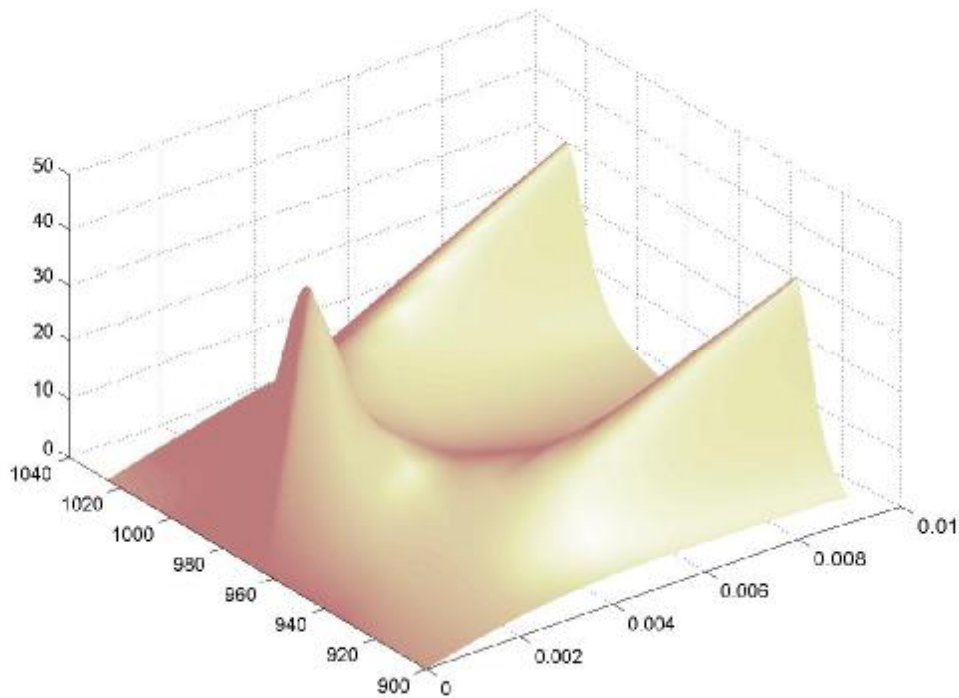


aunque se pueden realizar gráficas más sofisticadas:

```

>>t=0:0.001:0.009;
>>v=900:1025;
>>[T V]=meshgrid(t,v);
>>aux1=16*pi^2*(T.^2).*((V-918).^2).*((V-1011).^2);
>>aux2=aux1+(2*V-1929).^2;
>>w=T./aux2;
>>z=35000000*w;
>>surf1(t,v,z); % Este comando dibuja la superficie creada
mediante las
>>shading interp; % ordenes anteriores. Los siguientes sirven para
modificar
>>colormap(pink); % el dibujo obtenido
>>rotate3d; % Sirve para girar la figura mediante el ratón

```



PROGRAMACIÓN CON MATLAB

Para escribir un programa con MATLAB habrá que crear un fichero que tenga extensión `.m` y contenga las instrucciones. Esto se puede hacer con cualquier editor de textos, pero tiene algunas ventajas usar el editor propio de MATLAB llamado `medit`.

MATLAB trabaja con memoria dinámica, por lo que no es necesario declarar las variables que se van a usar. Por esta misma razón, habrá que tener especial cuidado y cerciorarse de que entre las variables del espacio de trabajo no hay ninguna que se llame igual que las de nuestro programa (proveniente, por ejemplo, de un programa previamente ejecutado en la misma sesión), porque esto podría provocar conflictos. A menudo, es conveniente reservar memoria para las variables (por ejemplo, si se van a utilizar matrices muy grandes); para ello, basta con asignarles cualquier valor. Del mismo modo, si se está usando mucha memoria, puede ser conveniente liberar parte de ella borrando (`clear`) variables que no se vayan a usar más.

Un programa escrito en MATLAB admite la mayoría de las estructuras de programación al uso y su sintaxis es bastante estándar. En los siguientes ejemplos se muestra la sintaxis de algunas de estas estructuras (`if`, `for`, `while`,...).

Ejemplo 1: Calcular la suma de los n primeros términos de la sucesión 1, 2x, 3x², 4x³, ...

```
n=input('¿Cuántos términos quieres sumar? ');
x=input('Dame el valor del numero x ');
suma=1;
for i=2:n
    suma=suma+i*x^(i-1);
end
disp('El valor pedido es')
disp(suma)
```

Ejemplo 2: Decidir si un número natural es primo.

```
n=input('Número natural que deseas saber si es primo ');
i=2;
primo=1;
while i<=sqrt(n)
    if rem(n,i)==0 % Resto de dividir n entre i
        primo=0;
        break
    end
    i=i+1;
end
if primo
    disp('El número dado es primo.')
else
    disp('El número dado no es primo.')
    disp('De hecho, es divisible por:')
    disp(i)
end
```

Ejemplo 3: Escribir un número natural en una base dada (menor que diez).

```
n=input('Dame el número que quieres cambiar de base ');
base=input('¿En qué base quieres expresarlo? ');
i=1;
while n>0
    c(i)=rem(n,base);
    n=fix(n/base); % Parte entera de n/base
    i=i+1;
end
disp('La expresión en la base dada es:')
i=i-1;
disp(c(i:-1:1))
```

Por último, también pueden programarse funciones. La primera instrucción de un fichero que contenga una función de nombre fun debe ser:

```
function [argumentos de salida]=fun(argumentos de entrada)
```

Es conveniente que el fichero que contenga la función se llame como ella; así, la función anterior debería guardarse en el fichero fun.m; por ejemplo, si se desea programar una función que calcule, mediante el algoritmo de Euclides,

el máximo común divisor de dos números naturales, basta escribir un fichero `euclides.m` cuyo contenido sea:

```
function m=euclides(a,b)
% Cálculo del máximo común divisor de dos números naturales
% mediante el algoritmo de Euclides
if a<b
    c=b;
    b=a;
    a=c;
end
while b>0
    c=rem(a,b);
    a=b;
    b=c;
end
m=a;
```

Si, una vez escrito el fichero anterior, en el espacio de trabajo o en un programa se escribe la instrucción

```
mcd=euclides(33,121)
```

en la variable `mcd` se almacenará el valor 11.

Las variables de una función son siempre locales. Por tanto, aunque en el seno de la función se modifiquen los argumentos de entrada, el valor de las variables correspondientes queda inalterado. Por ejemplo, en la función `euclides.m` se modifica el valor de los argumentos de entrada, pero, sin embargo:

```
>>x=15;
>>mcd=euclides(x,3);
>>x
x =
15
```

Si se pretende que las modificaciones de un argumento de entrada afecten a la variable correspondiente, deberá situarse dicho argumento, además, en la lista de argumentos de salida.